THESIS

DELAYING A CONVOY

by

Dong Hwan Oh

December 2005

Thesis Advisor:	R. Kevin Wood
Thesis Co-Advisor:	Sang Heon Lee
Second Reader:	Johannes O. Royset

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2005	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: Delaying a Convoy	**5. FUNDING NUMBERS**
6. AUTHOR(S)	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (maximum 200 words)

This thesis studies "the convoy-path interdiction problem" (CPIP) in which an interdictor uses limited resources to attack and disrupt road segments ("arcs") or road intersections ("nodes") in a road network in order to delay an adversary's convoy from reaching its destination. The convoy will move between a known origin node *a* and destination node *b* using a "quickest path."

We first show how to compute, using an A* search, the convoy's quickest path under the assumptions that the convoy may occupy several arcs simultaneously, each arc may have a different speed limit, and the convoy maintains constant inter-vehicle spacing. The basic model assumes that the convoy moves in a single lane of traffic; an extension handles arcs that may have multiple lanes. Using that algorithm as a subroutine, a decomposition algorithm solves the optimal interdiction problem. Interdiction of a node or arc makes that node or arc impassable.

Computational results are presented on grid networks with up to 629 nodes and 2452 arcs with varying levels of interdiction resource. Using Xpress-MP optimization software and a 2 GHz Pentium IV computer, the largest network problem solves in no more than 360 seconds given that at most 4 arcs can be interdicted.

14. SUBJECT TERMS Convoy Quickest Path, Network Interdiction, A* search, Resource allocation, Convoy path	**15. NUMBER OF PAGES** 69
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

DELAYING A CONVOY

Dong Hwan Oh
Captain, Republic of Korea Army
B.S., Korea Military Academy, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
December 2005

Author:	Dong Hwan Oh
Approved by:	Dr. R. Kevin Wood Thesis Advisor
	Dr. Sang Heon Lee Thesis Co-Advisor
	Dr. Johannes O. Royset Second Reader
	Dr. James N. Eagle Chairman, Department of Operations Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis studies "the convoy-path interdiction problem" (CPIP) in which an interdictor uses limited resources to attack and disrupt road segments ("arcs") or road intersections ("nodes") in a road network in order to delay an adversary's convoy from reaching its destination. The convoy will move between a known origin node *a* and destination node *b* using a "quickest path."

We first show how to compute, using an A* search, the convoy's quickest path under the assumptions that the convoy may occupy several arcs simultaneously, each arc may have a different speed limit, and the convoy maintains constant inter-vehicle spacing. The basic model assumes that the convoy moves in a single lane of traffic; an extension handles arcs that may have multiple lanes. Using that algorithm as a subroutine, a decomposition algorithm solves the optimal interdiction problem. Interdiction of a node or arc makes that node or arc impassable.

Computational results are presented on grid networks with up to 629 nodes and 2452 arcs with varying levels of interdiction resource. Using Xpress-MP optimization software and a 2 GHz Pentium IV computer, the largest network problem solves in no more than 360 seconds given that at most 4 arcs can be interdicted.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

ACKNOWLEDGMENTS

This thesis could not have been completed without the enormous support of these people. First of all, I would like to extend my deepest appreciation to my thesis advisor, Professor Kevin Wood. I am grateful for his countless hours of work, timely revisions, and guidance throughout the entirety of the thesis process. The advice and direction of co-advisor, KNDU Professor Lee, Sang Heon and Second Reader, Professor Johannes Royset allowed me to complete this thesis.

I would also like to thank my family (grandmother, parents, brothers and sister) for supporting me from Korea. They have been the greatest supporters and inspirers whenever I was discouraged.

I am also grateful for all Korean officers at NPS, brothers and sisters in the Young-Nak Church and all my American friends, especially Rat, Greg, Richard and Bill, and all the individuals and organizations who have helped me.

Additionally, I greatly appreciate my country, the Republic of Korea, its Army and KNDU for their support during my time at the Naval Postgraduate School.

Finally, thank you really so much God!!!

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis studies "the convoy-path interdiction problem" (CPIP) in which an interdictor uses limited resources to attack and disrupt road segments ("arcs") or road intersections ("nodes") in a road network in order to delay an adversary's convoy from reaching its destination. The convoy will move between a known origin node *a* and destination node *b* using a "quickest path." The quickest *a*-*b* path requires the least amount of time for the full convoy to move from *a* to *b*, and takes into account the size of the convoy, and each road segment's capacity (effective speed limit) and length. The quickest path for a convoy may be quite different from the path that would be computed with a standard shortest-path algorithm. That path, computed using arc transit times in place of arc lengths, would only be appropriate for a single, unaccompanied vehicle.

The convoy is assumed to move in an "open-column formation" and maintains a constant inter-vehicle spacing of 100 meters. If we suppose a battalion-to-brigade sized convoy consisting of 60-400 vehicles, its length may exceed 40 km. And, if we assume that typical road segments extend 10-50 km, it is clear that the convoy may occupy up to five road segments simultaneously. A standard shortest-path algorithm cannot identify the quickest path because the fact that the convoy must maintain constant inter-vehicle spacing implies that the convoy may not be able to use each road segment's full capacity (effective speed limit). In particular, the convoy's speed depends on the lowest speed limit on the road segments it occupies at any given time. For simplicity, we assume perfect coordination of the convoy so that it always moves as fast as possible. We also ignore slowdowns that might arise from turning corners, but "turn constraints" could be added to our formulations, at least approximately, through simple modifications of the road-network model.

We find no evidence that the quickest-path problem for a convoy has been modeled appropriately in the literature. Therefore, this thesis begins by devising an implicit path-enumeration algorithm for solving this problem. (This algorithm may be classified as a type of "A* search," or "branch-and-bound algorithm.") The basic model assumes that the convoy moves in “single file”; an extension allows the convoy to use as many traffic lanes as are available.

We then use the quickest-path algorithm as a subroutine in a known decomposition algorithm to solve the optimal, budget-limited interdiction problem. Interdiction of an arc or node makes that arc or node impassable, and that interdiction consumes some given portion of a fixed budget.

Computational results are presented on networks with up to 629 nodes and 2452 arcs with varying levels of interdiction resource. Using Xpress-MP optimization software and a 2 GHz Pentium IV computer, the largest network problem solves in no more than 360 seconds given that at most 4 arcs can be interdicted and made impassable.

I. INTRODUCTION

This thesis studies "the convoy-path interdiction problem" (CPIP) in which an interdictor uses limited resources to attack and disrupt segments ("arcs") or intersections ("nodes") in a road network in order to delay an adversary's convoy from reaching its destination. The convoy will move between known nodes, origin *a* and destination *b*, using a "quickest path." The quickest *a*-*b* path requires the least amount of time for the full convoy to move from *a* to *b*, and takes into account the size of the convoy, and each segment's length and "capacity," which is the effective speed limit. Interdiction of a node or arc is assumed to make that part of the road network impassable.

The convoy is assumed to move in "open-column formation" and to maintain a constant inter-vehicle spacing of 100 meters, or some other distance defined by the convoy's commander [FM 4-01-011 2002, Appendix C]. The convoy's total length may exceed 40 km. Given that the convoy can occupy several road segments simultaneously, and given constant inter-vehicle spacing, it is clear that the convoy's quickest *a*-*b* path is more complicated to compute than a standard shortest *a*-*b* path. Thus, CPIP looks like a standard "shortest-path network interdiction problem," (Israeli and Wood, 2002), except that "shortest path" is replaced by "quickest path," defined and computed by other means.

Fulkerson and Harding (1977) create the first shortest-path interdiction model. The interdictor has limited interdiction resource with which to maximize the length of a network user's shortest *a*-*b* path. The authors assume that the length of any arc increases linearly with the amount of a single interdiction resource applied. The resulting model converts into a simple, parametric linear-programming problem.

Ball, Golden and Vohra (1989) consider a discrete version of Fulkerson and Harding's problem in which the interdictor has a budget of *k*, the interdiction of an arc requires one unit of resource, and fractional interdictions are not allowed. This is the "k-most-vital-arcs problem." They solve this problem by using binary variables to determine whether an arc is destroyed or not.

Israeli and Wood (2002) generalize the k-most-vital arcs problem to allow general interdiction-resource constraints. They show how to solve this problem, denoted

"MXSP," with several primal decomposition schemes including Benders decomposition, a decomposition using a set-covering master problem, and a hybrid of those two.

Israeli et al. (2004) realize that MXSP is not an appropriate model for delaying a convoy, so they create a version of CPIP which looks like MXSP except that the quickest path for the convoy is computed using simple continuum traffic theory (SCTT) (e.g., Kuhne and Michalopoulos 1998). They solve this version of CPIP, successfully, by replacing the shortest-path subroutine with a quickest-path subroutine in one of the decomposition algorithms for MXSP. However, the SCTT model does not describe convoy movement with constant inter-vehicle spacing: According to their model, the convoy may compress or stretch as it traverses a path with varying arc capacities.

This thesis solves a more realistic version of CPIP by using a better quickest-path model, denoted "CQP-CS" (Convoy Quickest Path – Constant Spacing). We first show how to compute the convoy's quickest path under the assumption that the convoy may occupy several arcs simultaneously, each arc may have a different speed limit, and the convoy maintains constant inter-vehicle spacing. (This implicit path enumeration may be viewed as a version of "A* search," e.g., Russell and Norvig 1995, pp. 92-107.) The speed of the convoy is assumed to be the minimum of the speed limits that some vehicle in the convoy must obey. For simplicity, we assume perfect coordination among the vehicles in the convoy, no slowing for turns, and instantaneous acceleration and deceleration. (Note that "turn constraints" are easily approximated through modifications of the road-network model, e.g., Caldwell 1961.) The basic model also assumes that the convoy moves in a single lane of traffic, but a simple extension handles arcs with multiple lanes. Using that algorithm as a subroutine to solve CQP-CS, we solve CPIP using a decomposition algorithm developed by Israeli and Wood (2002).

This thesis is organized as follows: Chapter II provides background on the problem of computing a convoy's quickest path, compares alternatives, and describes an algorithm for solving a CQP-CS problem. Chapter III provides background on the problem of interdiction a convoy's path (CPIP), compares alternative models, and describes a generic algorithm. Chapter IV provides computational results, and Chapter V presents conclusions.

II. A CONVOY'S QUICKEST-PATH

This chapter provides background on quickest-path models for a convoy, describes the model we will use, and devises an implicit-path-enumeration algorithm to solve that model's instances.

A. BACKGROUND ON CONVOY MOVEMENT

We imagine a convoy that must move, as quickly as possible, between nodes *a* and *b* in a road network. What is the quickest path? Because a convoy can be quite long and must move according to doctrine, this "convoy quickest path problem" (CQPP) is more complicated than a standard shortest-path problem. To identify a reasonable model for this problem, we must first determine what a convoy actually looks like.

A convoy is organized by three elements which are a "march column," a "serial," and a "march unit." A march unit is the smallest subgroup of a convoy and usually consists of 15-20 vehicles. A serial consists of two to four march units and a march column consists of two to five serials. [FM 4-01-011 2002, Appendix C]. The march column makes up the full convoy. Table 1 specifies the dimensions of a convoy, and its subdivisions, for a brigade to battalion-sized element. If the convoy moves in the common "open-column formation," it will maintain an inter-vehicle spacing of 100 meters [FM 4-01-011 2002, Appendix C], and may thus have a length of up to 42 km.

	Lower Bound	Upper Bound
March Column	2 Serials	5 Serials
Serial	2 Marches	4 Marches
March Unit	15 Vehicles	20 Vehicles
Total Number of Vehicles	60 Vehicles	400 Vehicles
Vehicle Length	5 m	5 m
Vehicle Interval (Open Column)	100 m	100 m
Total Convoy Length	6.2 km	42 km

Table 1. A Battalion-to-Brigade Sized Convoy

If we assume that a convoy may be as long as 42 km and that road segments are typically 10-50 km in length, we see that the convoy may occupy up to five road segments simultaneously.

Given that the convoy can occupy several road segments simultaneously, and the convoy must maintain constant inter-vehicle spacing, it is clear that the convoy's quickest *a*-*b* path is more difficult to compute than a standard shortest *a*-*b* path.

B. QUICKEST-PATH MODELS FOR CONVOYS

We model the road network as a directed graph $G = (N, A)$ where N is a set of nodes and A is a set of arcs. An arc is an ordered pair of nodes (i, j) where $i, j \in N$. Nodes represent intersections or interchanges in the road network, the arcs represent road segments. The convoy must move from node *a* to node *b*. For simplicity, we assume the head of the convoy is at *a* at the start of the convoy's movement, and the transit to *b* is not complete until the last vehicle in the convoy has cleared *b*.

Each arc (i, j) has a *length* l_{ij} measured in kilometers (km), a *speed limit* s_{ij} measured in kilometers per hour (km/hr), and the *number of lanes* available for use c_{ij}. Note that the convoy's commander may decide to move in single file, in which case c_{ij} is always 1, or he (or she) may decide to use all lanes available up to 4, or may use some other guidelines. In any case, c_{ij} is always known. The transit time for the convoy along any path depends on all of those values, in addition to the convoy's length, L (km).

1. Convoy Quickest Paths as Shortest Paths (CQP-SP)

We can approximate the movement of a convoy as a single vehicle moving along a standard shortest path where the "length" of each arc (i, j) is its traversal time $t_{ij} = l_{ij} / s_{ij}$. This will lead to a very efficient method for finding a "quickest path," but it may not represent the movement of a convoy accurately.

If A_p denotes the ordered set of arcs in the shortest path, the convoy's total path-transit time is estimated as

$$z_P = \sum_{(i,j)\in A_p} t_{ij} + L/s'$$

where s' is the speed of the last arc in A_P. Thus, the term L/s' estimates the clearing time for the convoy once its head reaches *b*. The convoy's quickest path under this model is approximated by solving a shortest-path problem between *a* and *b* with respect to arc "lengths" t_{ij} and by then identifying s' and adding L/s'.

2. Convoy Quickest Paths by Simple Continuum Traffic Theory (CQP-SCTT)

In solving their version of CPIP, Israeli et al. (2004) use a CQP model based on "simple continuum traffic theory" (SCTT). We label this model as CQP-SCTT. SCTT represents a stream of traffic as a fluid in which the flow rate cannot exceed the maximum capacity of a given arc. Because of this model's orientation toward "flow," its data are defined differently than ours, but appropriate conversions will be given.

Let f_{ij} denote the flow rate (capacity) on arc (*i*, *j*) in vehicles per hour. SCTT represents a bottleneck as a junction between two arcs, (i, j) and (j, k), where f_{ij} exceeds f_{jk}. The upstream mass on (*i, j*) flowing at a rate greater than f_{jk} is forced to decrease its speed and flow rate, and the flow (traffic) increases in density on arc (*i, j*). Meanwhile, the downstream mass continues to flow through (*j, k*) at rate f_{jk}. Whenever the bottleneck is encountered in a path, the maximum flow rate of the path is controlled by the arc with the minimum capacity (Lighthill and Whitham 1955). The data for this model are:

t_{ij}	travel time for a single vehicle traversing (*i, j*), $t_{ij} = l_{ij}/s_{ij}$, (hours),
f_{ij}	peak rate of traffic flow over arc (*i, j*) (vehicles per hour), and
S	size of convoy (vehicles)

The transit time for a path A_P is therefore computed as

$$z_p = S/f_p + \sum_{(i,j)\in A_p} t_{ij}\text{, where } f_p = \min_{(i,j)\,\in A_p} f_{ij}\,,$$

and thus the quickest-path time is simply

$$z^* = \min_{p\in P}\left\{S/f_p + \sum_{(i,j)\,\in A_p} t_{ij}\right\}.$$

The flow rate f_{ij} defines the effective speed of the "fluid" moving through (*i, j*), so we can compute this value from our basic data as $f_{ij} = s_{ij}c_{ij}$. Additionally, we define a constant, say α, that converts convoy length size *S*, i.e., total number of vehicles. In our case, $\alpha = 1 \text{ vehicle} / 0.105 \text{ km}$ α has to be computed from the numerical data which implies that one vehicle takes up 105 meters, i.e., 0.105 km. So, it has 1/0.105 vehicles per kilometer as a conversion factor. Therefore, we can convert as follows:

$$z^* = \min_{p\in P}\left\{S/f_p + \sum_{(i,j)\,\in A_p} t_{ij}\right\} = \min_{p\in P}\left\{\alpha L/\{(sc)_p\} + \sum_{(i,j)\,\in A_p} t_{ij}\right\},\ \text{where } (sc)_p = \min_{(i,j)\,\in A_p} s_{ij}c_{ij}\,.$$

We use A* search, based on this equation, to solve the CQP-SCTT in Chapter IV.

The CQP-SCTT model violates our assumption of constant inter-vehicle spacing, however. In particular, spacing is reduced on arcs with low f_{ij} arcs and increased on arcs with high f_{ij}. As the head of a convoy moves from arc (i,j) with high f_{ij} arc to (j,k) with low f_{jk}, vehicles bunch up, becoming easier targets for the enemy; when the transition is from low capacity to high capacity, the vehicles in the lead will "race ahead," creating large inter-vehicle spaces which create the opposite problem: Vehicles may be too far apart to help defend each other. The CQP-SCTT model may be appropriate for a group of vehicles moving independently along a single path, but it is inappropriate for a convoy in which vehicles move in a coordinated fashion.

3. Convoy Quickest Paths with Constant Inter-vehicle Spacing (CQP-CS)

We wish to create a more appropriate quickest-path model for a convoy. We first note that the convoy may be many kilometers long, and may therefore cover several different road segments, each with its own length and speed limit. Given constant inter-vehicle spacing, it is easy to see that the convoy's speed is dictated by the slowest road segment the convoy currently occupies. The real-world problem is somewhat more complicated because vehicles do not change speeds instantaneously and communications may be imperfect, but, for simplicity, we assume that the full convoy will instantaneously change speed, as necessary, to match the speed of the slowest segment it occupies. Our solution methodology is quite flexible, and can easily accommodate variations such as different acceleration and deceleration rates.

Figure 1 illustrates the basic concept of computing total path-transit time from node a to node b. We assume that the convoy is initially lined up on an artificial arc (a', a) of length L; this arc has an arbitrarily high speed limit. The convoy finishes its a-b transit only when its last vehicle has reached b, i.e., the whole convoy has reached and cleared b. We model this by creating an artificial arc (b, b') having length L and an arbitrarily high speed limit, and by then defining the a-b transit to be complete when the head of the convoy reaches b'. The total transit time for a given path is broken down into the times T_{ij} that are required for the head of the convoy to move from node i to node j on arc (i, j). Computing each such T_{ij} involves determining how the set of arcs the convoy is in contact with changes as its head moves forward.

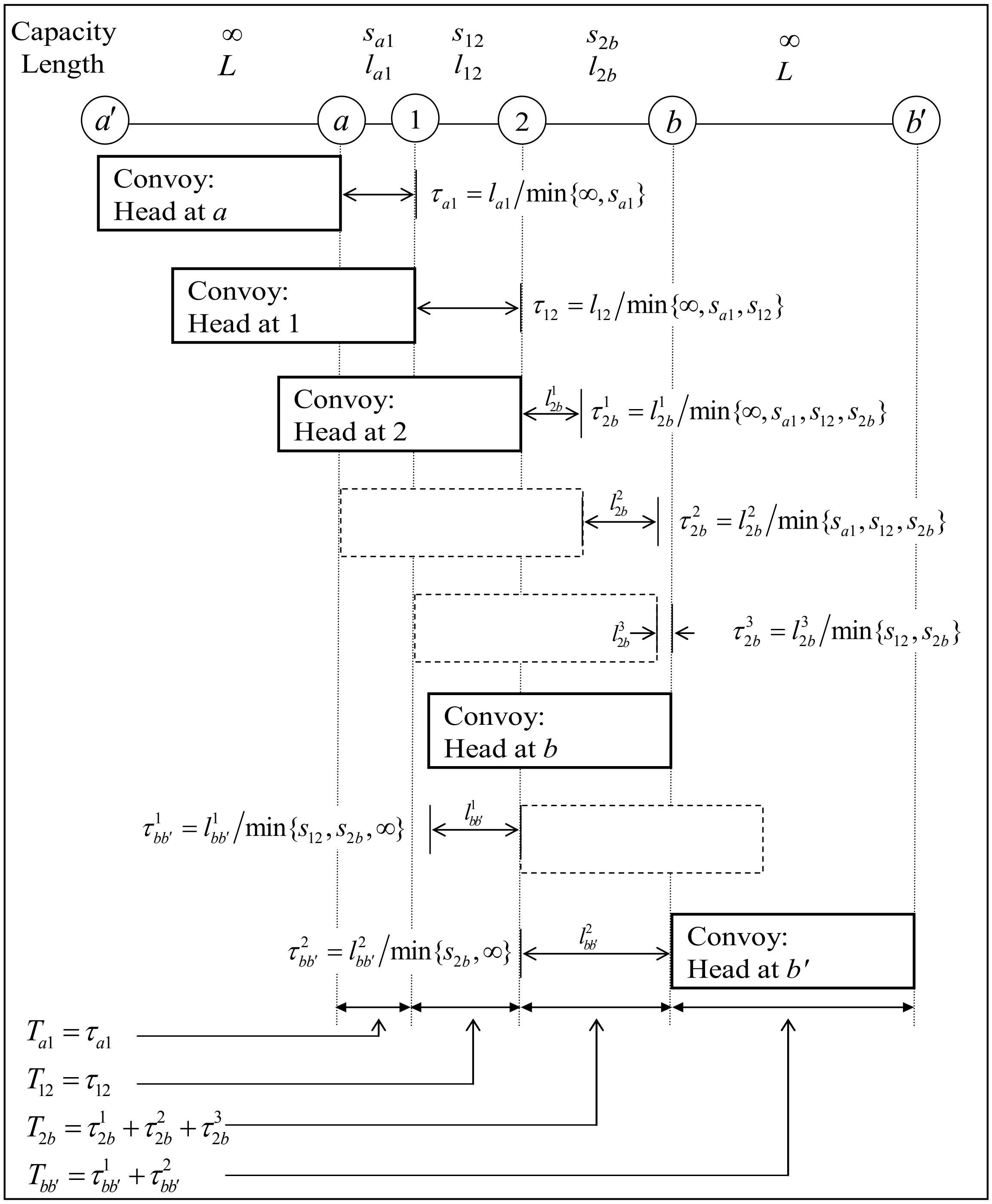

Capacity
Length
∞
L
s_{a1}
l_{a1}
s_{12}
l_{12}
s_{2b}
l_{2b}
∞
L
a'
a
1
2
b
b'
Convoy: Head at a
$\tau_{a1} = l_{a1}/\min\{\infty, s_{a1}\}$
Convoy: Head at 1
$\tau_{12} = l_{12}/\min\{\infty, s_{a1}, s_{12}\}$
Convoy: Head at 2
l_{2b}^1
$\tau_{2b}^1 = l_{2b}^1/\min\{\infty, s_{a1}, s_{12}, s_{2b}\}$
l_{2b}^2
$\tau_{2b}^2 = l_{2b}^2/\min\{s_{a1}, s_{12}, s_{2b}\}$
l_{2b}^3
$\tau_{2b}^3 = l_{2b}^3/\min\{s_{12}, s_{2b}\}$
Convoy: Head at b
$\tau_{bb'}^1 = l_{bb'}^1/\min\{s_{12}, s_{2b}, \infty\}$
$l_{bb'}^1$
$\tau_{bb'}^2 = l_{bb'}^2/\min\{s_{2b}, \infty\}$
$l_{bb'}^2$
Convoy: Head at b'
$T_{a1} = \tau_{a1}$
$T_{12} = \tau_{12}$
$T_{2b} = \tau_{2b}^1 + \tau_{2b}^2 + \tau_{2b}^3$
$T_{bb'} = \tau_{bb'}^1 + \tau_{bb'}^2$

Figure 1. Illustration of a convoy's transit time through a simple path in a network. Tij represents the transit time for the convoy's head to move from node i to node j. Total transit times may consist of the sum of several "τ variables," each representing the time the convoy is subject to the speed limits on a given set of arcs. The overall speed limit for the convoy at any time is the minimum of the limits any vehicle in the convoy is subject to. The total transit time here is Ta1 + T12+ T2b+ Tbb′ .

Algorithm 1, below, implements the ideas illustrated in Figure 1, and discussed above, to compute the transit time for a given *a*-*b* path with arc set A_P.

Definitions for Algorithm 1:

Sets, Indices and Structural Data

$G = (N, A)$	directed graph with node set *N* and arc set *A*
$a, b \in N$	origin and destination nodes, respectively
A_P	ordered list of arcs on the *a*-*b* path *P*, $A_P = \{(i_1, i_2), (i_2, i_3), \ldots, (i_{n_p}, i_{n_p+1})\}$

Numerical Data [units]

L	convoy length [km]
l_{ij}	length of arc $(i, j) \in A$ [km]
s_{ij}	capacity (speed) of arc $(i, j) \in A$ [km/hour]

Variables	**(including sets)**
T	total transit time for convoy [hour]
A'	set of all arcs occupied currently by convoy, $A' = \{(i, j)^{back}, (i, j)^1, (i, j)^2, \ldots, (i, j)^{front}\}$
h	counter for arc on the path A_P
δ_{front}	distance convoy has moved along the arc at the convoy's front
δ_{back}	distance convoy has moved along the arc at the convoy's back
δ_{next}	incremental distance the convoy will move next

Algorithm 1: Compute convoy transit time for a given path assuming constant inter-vehicle spacing.

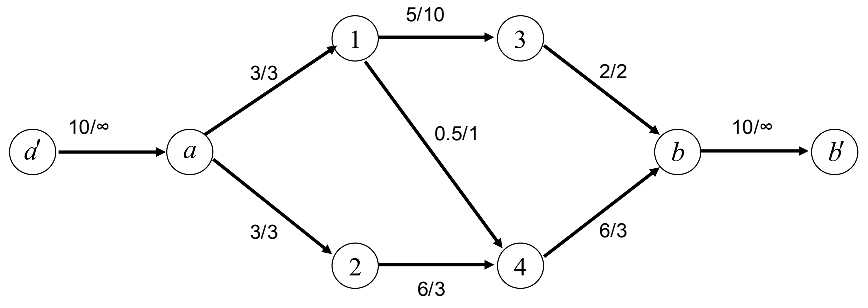
Input: A directed graph $G = (N, A)$ with source node *a,* sink node *b*, arc lengths $l_{ij} \geq 0$ and arc speeds $s_{ij} \geq 0$ $\forall (i, j) \in A$, convoy length *L*, an ordered list of arcs A_P arcs that defines an *a*-*b* path.

Output: Transit time on *a*-*b* path A_P for the convoy. This time includes the time required to completely clear *b*.

```
{
    /* Add artificial nodes and arcs to $G$ */
    $s_{\max} \leftarrow \max_{(i,j)\in A} s_{ij}$;
    $N \leftarrow N \cup \{a', b'\}$;
    $A \leftarrow A \cup \{(a', a), (b, b')\}$;
    $l_{a'a} \leftarrow L$;  $l_{bb'} \leftarrow L$;
    $s_{a'a} \leftarrow s_{\max}$;  $s_{bb'} \leftarrow s_{\max}$;
    /* Initialize */
    $T \leftarrow 0$;
    $A' \leftarrow \{(a', a)\}$;
    $h \leftarrow 0$;  /* arc at front of convoy will be $(i, j)_h$ */
    $\delta_{\text{back}} \leftarrow 0$;
    /* Main Algorithm */
    while ( $A' \neq \{(b', b)\}$ ) {
        $h \leftarrow h + 1$;
        Append $(i, j)_h$ to $A'$;
        $\delta_{\text{front}} \leftarrow 0$;  /* distance along $(i, j)_h$ the convoy has moved */
        while( $\delta_{\text{front}} < l_{ij}$ ) {
            $s_{\min} \leftarrow \min_{(i,j)\in A'} s_{ij}$;
            /* $(i', j') = (i, j)^{back}$ and $(i'', j'') = (i, j)^{front}$ in $A'$ */
            $\delta_{\text{next}} \leftarrow \min\{l_{i'j'} - \delta_{back}, l_{i''j''} - \delta_{front}\}$;
            $T \leftarrow T + \delta_{\text{next}} / s_{\min}$;
            $\delta_{\text{front}} \leftarrow \delta_{\text{front}} + \delta_{\text{next}}$;  $\delta_{\text{back}} \leftarrow \delta_{\text{back}} + \delta_{\text{next}}$;
            if ( $\delta_{\text{back}} = l_{i'j'}$ ){
            /* the last transition moved the convoy's tail off of $(i, j)^{\text{back}}$ */
            delete $(i, j)^{\text{back}}$ from $A'$ and define a new $(i, j)^{\text{back}}$;
            $\delta_{\text{back}} \leftarrow 0$;
        }
    }
    /* after ending above "while-loop", $T$ is the same as $T_{ij}$ in Figure 1. */
    Print ( "Total transit time for the convoy is ", $T$, " hours.");
}
```

4. A Comparison of Quickest-path Models

Now that we have defined three alternatives for estimating a convoy's transit time, we can illustrate the differences. Figure 2 shows a small network with arcs having a single lane. Speed limits and lengths are specified for each arc. CQP-CS identifies Path 3 as the quickest path, CQP-SCTT identifies Path 1, and CQP-SP identifies Path 2. The estimated transit times for the three paths can be quite different. We have argued that the CQP-CS model is more appropriate, and if we assume it is correct (even though it is oversimplified), it can be seen that the estimated transit times for the other two models can underestimate the true transit time substantially. The example illustrates that the CQP-SP and CQP-SCTT models are not appropriate for describing convoy movement realistically and accurately, and CQP-CS is required.



- Arcs are marked as length (km) / speed (km/h)
- All arcs have a single lane.
- Convoy length is 10 km

Path	Estimated Travel Time (hours)		
	CQP-SP	CQP-SCTT	CQP-CS
1 : *a*-1-3-*b*	7.50	**7.50**	8.67
2 : *a*-1-4-*b*	**6.83**	13.50	13.50
3 : *a*-2-4-*b*	8.33	8.33	**8.33**

Figure 2. An example of different quickest-paths and their estimated transit times. Bold times indicate the quickest path identified for the given model. Note how the SP and SCTT models can substantially underestimate the "true" transit time as defined by the CS model.

C. SOLVING BASIC MODEL OF CQP-CS

We use an A* search to compute the solution to the CQPP. To make this algorithm easier to understand, we first describe a simpler A* search that solves a "nonlinear shortest-path problem" for a single vehicle.

1. General A* Algorithm for single vehicle shortest-path

We want to compute $\min_P f(A_p)$, the quickest *a*-*b* path for a single vehicle in *G*, where $f(\cdot)$ is a known nonlinear function with certain properties. In particular, each arc $(i,j)\in A$ has nominal transit time $t_{ij}\geq 0$, but $f(A_P)\geq \sum_{(i,j)\in A_P} t_{ij}$, i.e., the "true length" of a directed, simple path (between any pair of nodes) exceeds the sum of the path's arc lengths. If $F(i)$ denotes the true length of some *a*-*i* path A_i, i.e., $F(i)=f(A_i)$, and if $d(j)$ denotes the minimum transit time from *j* to *b* computed with respect to the t_{ij}, i.e., the standard shortest-path "length" using the t_{ij} as lengths, then we know that any complete *a*-*b* path A_p path that is an extension of A_i must satisfy $f(A_P) \geq F(i) + t_{ij} + d(j)$. This fact can be used in a standard A* search to compute $\min_P f(A_p)$, as follows.

Algorithm 2: General A* search for a single-vehicle, nonlinear, shortest-path problem

Solves $\min_P f(A_P)$ where A_P is an *a*-*b* path in network *G* and $f(A_P)$ is a given function that satisfies conditions described above.

Input: A directed graph $G = (N, A)$ with source node *a,* sink node *b*, arc lengths $l_{ij} \geq 0$ for all $(i, j) \in A$, arc speeds $s_{ij} > 0$ for all $(i, j) \in A$, for all $j \in N$, $d(j)$ computed as the minimum transit time from *j* to *b* with respect to function $f(A_i)$ that satisfies the conditions described above for any *a*-*i* path A_i.

Output: $\min_P f(A_P)$ and $\operatorname{argmin}_P f(A_P)$.

```
{
    /* Initialize */
    $UB \leftarrow \infty$;
    $A_P^* \leftarrow \varnothing$;  /* Best path found */
    $A_P \leftarrow \varnothing$;  /* Stack to keep track of the arcs in the current path */
    $N_P \leftarrow a$;  /* Stack to keep track of nodes on the current path */
    $F(a) \leftarrow 0$;   /* "length" of the current, null path $A_P$ from a to a */
    $t_{ij} \leftarrow l_{ij} / s_{ij}\ \forall (i, j) \in A$;
    for ( all $i \in N$ ) nextArc($i$) $\leftarrow$ firstArc($i$);
    isOnPath($a$) $\leftarrow$ true;
    for ( all $j \in N - a$ ) isOnPath($a$) $\leftarrow$ false;
    /* Main Algorithm */
    while ( $N_P \neq \emptyset$ ){
        $i \leftarrow$ node at top of $N_P$;
        if ( nextArc($i$) $\neq$ NULL) and $i \neq b$) ){
            $(i, j) \leftarrow$ arc pointed to by nextArc($i$);
            increment nextArc($i$);
            if ( $F(i) + t_{ij} + d(j) < UB$ ) { /* Test 1 */
                push $j$ onto $N_P$;
                push $(i, j)$ onto $A_P$;
                $F(j) \leftarrow f(A_P \cup \{(i, j)\})$;
                if ( $F(j) + d(j) < UB$ ) { /* Test 2 */
                    isOnPath($j$) $\leftarrow$ true;
                    $i \leftarrow j$;
                } else {
                    pop $N_P$;
                    pop $A_P$;
                }
            }
        }
        if ( $i = b$ )    $A_P^* \leftarrow A_P$; /* Best path found so far */
        isOnPath($i$) $\leftarrow$ false;
        nextArc($i$) $\leftarrow$ firstArc($i$);
        pop $N_P$;
        pop $A_P$;
    }
    print("Minimum time of an $a$-$b$ path is ", $UB$, "hours.");
    print("Optimal $a$-$b$ path is ", $A_P^*$);
}
```

2. Basic A* Search for the Convoy Quickest-Path Problem (CQP-CS)

We now modify the A* search algorithm above to solve the CQP-CS. In essence, the new algorithm, Algorithm 3, incorporates Algorithm 1 into the A* search of Algorithm 2. It is assumed that the convoy moves in a single lane of traffic.

To construct Algorithm 3, we first define $F(i)$ in Algorithm 2 as the time required by the head of convoy to reach node *i* using the current *a-i* path A_i (as illustrated in Figure 1 and as defined precisely in Algorithm 1). Then, we define $d(j)$, for all $j \in N$, as a lower bound on the time required for a convoy whose head is at *j* to reach and clear *b*. A reasonable lower bound is the minimum transit time with respect to $t_{ij} = l_{ij} / s_{ij}$, from *j* to *b*, plus some lower bound on the amount of time the convoy will require to clear *b*. Such a lower bound is $t' = L / \max_{(i,j)\in A|j=b} s_{ij}$. Thus, $d(j)$ can be computed by running a single, backward shortest-path algorithm starting at *b*, using edge lengths $t_{ij} = l_{ij} / s_{ij}$, and adding t' to the calculated minimum transit time at each node *j*. Given these definitions, the Algorithm-3 analogs of Test 1 and Test 2 from Algorithm 2 will be valid, and the full algorithm will be valid.

Some additional notation for Algorithm 3 is:

$A'(\cdot)$	an array that gives the arcs on the current path to the head of the convoy, used to define that path and the convoy's location, using additional pointers, described below
pQfront	pointer to the arc at front of convoy on $A'(\cdot)$
pQback	pointer to the arc at back of convoy on $A'(\cdot)$
pQfrontAtNode(*i*)	array to track pQfront when the head of the convoy reaches node *i* on the current path A_P
pQbackAtNode(*i*)	array to track pQback when the head of the convoy reaches node *i*on the current path A_P
dBackAtNode(*i*)	array to track δ_{back} when the head of the convoy reaches node *i* on the current path A_P

Algorithm 3: Modified A* search for CQP-CS.

Input: A directed graph $G = (N,\ A)$ with source node *a,* sink node *b*, arc lengths $l_{ij} \geq 0$, arc speeds $s_{ij} > 0$, strict upper bound UB on minimum convoy transit time, $d(j)$ computed as the minimum transit time from *j* to *b* with respect to $t_{ij} = l_{ij} / s_{ij}$ plus a lower bound $t' = L / \max_{(i,j)\in A|j=b} s_{ij}$ on the clearing time at *b*.

Output: An *a*-*b* path that minimizes convoy transit time (with constant inter-vehicle spacing), and that time.

```
{
    /* Add artificial nodes and arcs to $G$ */
    $N \leftarrow N \cup \{a', b'\}$;
    $A \leftarrow A \cup \{(a', a), (b, b')\}$;
    $l_{a'a} \leftarrow L$;  $l_{bb'} \leftarrow L$;
    $s_{a'a} \leftarrow s_{\max}$;  $s_{bb'} \leftarrow s_{\max}$;
    /* Initialize */
    $A_P^* \leftarrow \varnothing$;  /* Best path found */
    $A'(1) \leftarrow (a', a)$;  /* Array to keep track of the path being traversed and the
                                    arcs currently occupied by convoy */
    $N_P \leftarrow a$;  /* Stack to keep track of nodes on the current path */
    $F(a) \leftarrow 0$;   /* “time” of the current, null path $A_P$ from $a$ to $a$ */
    pQfrontAtNode($a'$) $\leftarrow$ 1;
    pQbackAtNode($a'$) $\leftarrow$ 1; /* Points to arc at back of convoy on $A'(\cdot)$ */
    dBackAtNode($a'$) $\leftarrow$ 0;
    for ( all $i \in N$ ) nextArc($i$) $\leftarrow$ firstArc($i$);
    isOnPath($a$) $\leftarrow$ true;
    for ( all $j \in N - a$ ) isOnPath($a$) $\leftarrow$ false;
    /*Define $t_{ij}$ is the time for a single vehicle to traverse arc $(i, j)$ */
```

```
for ( all $(i,j) \in A$ ) $t_{ij} \leftarrow l_{ij} / s_{ij}$ ;
/* Main Algorithm */
while( $N_P \neq \emptyset$ ){
        $i \leftarrow$ node at top of $N_P$;
        while ( nextArc($i$) $\neq$ NULL and $i \neq b'$ ){
                $(i,j) \leftarrow$ arc pointed to by nextArc($i$);
                increment nextArc($i$) ;
                if ( $F(i) + t_{ij} + d(j) < UB$ and $not$ isOnPath($j$) ) { /* Test 1 */
                        pQfront $\leftarrow$ pQfrontAtNode($i$);
                        pQback $\leftarrow$ pQbackAtNode($i$);
                        $\delta_{\text{back}} \leftarrow$ dBackAtNode($i$);
                        pQfront $\leftarrow$ pQfront+1;
                        $A'$(pQfront) $\leftarrow (i,j)$; /*add arc($i, j$) to front of $A'$*/
                        $\delta_{\text{front}} \leftarrow 0$; $T \leftarrow 0$;
                        while ( $\delta_{\text{front}} < l_{ij}$ ) {
                                $s_{\min} \leftarrow \min_{(i,j)\in A'} s_{ij}$;
                        /* $(i',j')$ is a back arc of $A'$, $(i'',j'')$ is a front arc of $A'$ */
                                $\delta_{\text{next}} \leftarrow \min\{l_{i'j'} - \delta_{back}, l_{i''j''} - \delta_{front}\}$;   /* Step 1a */
                                $T \leftarrow T + \delta_{\text{next}} / s_{\min}$;                          /* Step 1b */
                                $\delta_{\text{front}} \leftarrow \delta_{\text{front}} + \delta_{\text{next}}$;                      /* Step 1c */
                                $\delta_{\text{back}} \leftarrow \delta_{\text{back}} + \delta_{\text{next}}$;                       /* Step 1d */
                                if ( $\delta_{\text{back}} = l_{i'j'}$ ){
                                pQback $\leftarrow$ pQback+1;  /*delete a back arc in $A'$*/
                                $\delta_{\text{back}} \leftarrow 0$;
                                }
                        }
                        $F(j) \leftarrow F(i) + T$;
```

```
                    if ( $F(j) + d(j) < UB$ ) {  /* Test 2 */
                        push $j$ onto $N_P$;
                        $i \leftarrow j$;
                        isOnPath($i$) $\leftarrow$ true;
                        pQfrontAtNode($i$) $\leftarrow$ pQfront;
                        pQbackAtNode($i$) $\leftarrow$ pQback;
                        dBackAtNode($i$) $\leftarrow \delta_{\text{back}}$;
                    }
                }
            }
            if ( $i = b'$ ) {
                $A_P^* \leftarrow \{A'(1),\ldots,A'(\text{pQfront})\}$;          /* Best path found so far */
                $UB \leftarrow F(i)$;
            }
            isOnPath($i$) $\leftarrow$ false;
            nextArc($i$) $\leftarrow$ firstArc($i$);
            pop $N_P$;
            pop $A'(\text{pQfront})$;
        }
        print("Minimum convoy transit time of an a-b path is ", UB, "hours.");
        print("Optimal a-b path is ", $A_P^*$);
}
```

3. Enhanced A* Algorithm for the Convoy Quickest-Path Problem

A* search is essentially a branch-and-bound algorithm, and such algorithms can often be improved by using a heuristic to identify a good starting solution and upper bound, *UB*. We use the obvious heuristic: Find the quickest path for a single vehicle using a shortest-path algorithm, and evaluate the true convoy-transit time for that path to yield UB. Rather than writing a special algorithm to compute that transit time, we simply use Algorithm 3 as a subroutine with modified data, as follows:

Algorithm 3E: Enhanced A* search for solving the convoy quickest-path problem (CQP-CS)

Input: Same as Algorithm 3.

Output: An *a*-*b* path that minimizes convoy traversal time (CS), and that time.

```
{
    Step 0:  $t_{ij} \leftarrow l_{ij} / s_{ij} \quad \forall (i, j) \in A$;  /* Initialize */
    Step 1:  Compute $d(i) \quad \forall i \in N$ as the shortest transit time for a single vehicle
              from $i$ to $b$ plus $t' = L / \max_{(i,j) \in A | j=b} s_{ij}$;
              /* Solve one backward shortest-path prob. with respect to $t_{ij}$ from $b$ */
              Identify arcs on shortest-path $A_s$;
    Step 2:  Modify data so that $t_{ij} \leftarrow l_{ij} \leftarrow \infty \quad \forall (i, j) \in A - A_s$;
    Step 3:  Call Algorithm 3 with modified $t_{ij}$, $UB = \infty$ and with modified
             Identify new $UB$;  /* Clearly, only the arcs in $A_s$ will be traversed */
    Step 4:  Reset original arc lengths and transit times $l_{ij}$ and
             $t_{ij} \leftarrow l_{ij} / s_{ij} \quad \forall (i, j) \in A$;
    Step 5:  Call Algorithm 3 with original data but with new $UB$, and
             initialize $A_P^*$ as $A_s$ ;
}
```

D. EXTENDING CQP-CS TO MULTIPLE LANES

The convoy quickest-path model in the previous section assumes that the convoy moves in a single lane of traffic. However, it is certainly possible that a convoy could make use of, say, three out of four lanes on a highway, if such were available. (The fourth lane might be set aside for repair and emergency vehicles.) This section develops a quickest-path model and an algorithm that lets the convoy use a variable number of traffic lanes defined by c_{ij}. These values can be fractional to represent inefficiencies in using extra lanes s_{ij}. For instance, vehicles will probably be staggered in the various lanes, and if each lane has an inter-vehicle spacing of 100 meters, the spacing that a strafing enemy aircraft might see would be less than 33 meters. Thus, a commander might actually increase inter-vehicle spacing beyond 100 meters and thereby lose some of the capacity advantage afforded by multiple lanes. For simplicity, we assume perfect coordination of lane changes so that no overhead is incurred.

Algorithm 3EM, which solves the multi-lane model, is specified below. It may be viewed as a simple variant of Algorithm 3 with a convoy whose length depends on the number of lanes it is operating on.

Algorithm 3EM: Multi-lane Convoy Quickest-Path Model

Solves "CQP-CSML," which is the same as CQP-CS except that multiple lanes of traffic may be used, if available.

Input: Same as Algorithm 3E except with the number of traffic lanes c_{ij} defined for each $(i,j) \in A$; Note that $c_{a'a} \equiv c_{bb'} \equiv 1$;

Output: An *a*-*b* path that minimizes convoy transit time (CS) when multiple lanes may be used, and that path's transit time.

```
{
    Step 0: $t_{ij} \leftarrow l_{ij} / (s_{ij} \times c_{ij}) \quad \forall (i,j) \in A$;    /* Initialize */
    Step 1: Compute $d(i) \quad \forall i \in N$ as the shortest transit time for a single vehicle
            from $i$ to $b$ plus $t' = L \Big/ \left( \max_{(i,j) \in A | j=b} s_{ij} c_{ij} \right)$;
            /* Solve one backward shortest-path prob. with respect to $t_{ij}$ from $b$ */
            Identify arcs on shortest-path $A_s$;
    Step 2: Modify data so that $t_{ij} \leftarrow l_{ij} \leftarrow \infty \quad \forall (i,j) \in A - A_s$;
    Step 3: Call Algorithm 3 with modified $t_{ij}$, $UB = \infty$ and with steps 1a – 1d
            replaced by;
            $\delta_{\text{next}} \leftarrow \min\{(l_{i'j'} - \delta_{back}) \times c_{i'j'}, (l_{i''j''} - \delta_{front}) \times c_{i''j''}\};$   /* Step 1a */
            $T \leftarrow T + \delta_{\text{next}} /(s_{\min} \times c_{i''j''});$   /* Step 1b */
            $\delta_{\text{front}} \leftarrow \delta_{\text{front}} + \delta_{\text{next}} / c_{i''j''};$   /* Step 1c */
            $\delta_{\text{back}} \leftarrow \delta_{\text{back}} + \delta_{\text{next}} / c_{i'j'};$   /* Step 1d */
        Identify new $UB$; /* Clearly, only the arcs in $A_s$ will be traversed */
    Step 4: Reset original arc lengths and transit times $l_{ij}$
        and $t_{ij} \leftarrow l_{ij} / (s_{ij} \times c_{ij}) \quad \forall (i,j) \in A;$
    Step 5: Call Algorithm 3 with original data but with new $UB$, and
        steps 1a – 1d replaced by;
            $\delta_{\text{next}} \leftarrow \min\{(l_{i'j'} - \delta_{back}) \times c_{i'j'}, (l_{i''j''} - \delta_{front}) \times c_{i''j''}\};$   /* Step 1a */
            $T \leftarrow T + \delta_{\text{next}} /(s_{\min} \times c_{i''j''});$   /* Step 1b */
            $\delta_{\text{front}} \leftarrow \delta_{\text{front}} + \delta_{\text{next}} / c_{i''j''};$   /* Step 1c */
            $\delta_{\text{back}} \leftarrow \delta_{\text{back}} + \delta_{\text{next}} / c_{i'j'};$   /* Step 1d */
        And with $A_s$ initialized by $A_P^*$;
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

III. CONVOY -PATH INTERDICTION

This chapter provides general background on "path-interdiction problems," defines the convoy-path interdiction problem precisely, and proposes basic and enhanced models for its solution.

A. BACKGROUND ON CONVOY-PATH INTERDICTION

This thesis describes the "convoy-path interdiction problem" (CPIP) in which an interdictor uses limited resources to attack and disrupt road segments (arcs) or road intersections (nodes) in a road network in order to delay an adversary's convoy from reaching its destination. We assume that the convoy commander has full knowledge of the state of the relevant road network after interdiction, and will move the convoy to its destination using use the quickest route available. That is, he will plan his route using one of the quickest-path algorithms described in the previous chapter. We also note that a node can be converted to an arc for the purposes of interdiction by using the standard technique of "node-splitting" (e.g., Ahuja et al. 1993, pp. 41-42), so we discuss interdiction of arcs only in the following.

Since CPIP is NP-hard (Israeli and Wood 2002) and evidently has no simple formulation as a mixed-integer program, we use the set-partitioning decomposition and algorithm of Israeli and Wood to identify an optimal interdiction plan, and then attempt to improve that technique using ideas from Benders decomposition. Before presenting algorithms, we require a few definitions, assumptions and notation:

1) If an arc is interdicted, it is completely destroyed and becomes impassable.

2) **x** denotes an interdiction plan (vector) such that

$$x_{ij} = \begin{cases} 1 & \text{if } (i,j) \text{ is interdicted,} \\ 0 & \text{otherwise.} \end{cases}$$

3) **r** denotes a vector of r_{ij} required resource units to interdict arc (i,j).

4) X denotes the set of feasible interdiction plans:

$$X = \left\{ \mathbf{x} \in \{0,1\}^{|A|} \mid \mathbf{rx} \le R \right\}$$

5) **y** denotes an arc-path incidence vector for an *a*-*b* path *p* in *G* such that

$$y_{ij} = \begin{cases} 1 & \text{if } (i,j) \in A_p, \\ 0 & \text{otherwise.} \end{cases}$$

6) *Y* denotes a set of arc-path incidence vectors.

7) The total amount of interdiction resource is *R,* each arc (i, j) requires r_{ij} units of resource to interdict. However, artificial arcs (a', a) and (b, b') cannot be interdicted, so we may assume $r_{a'a} = r_{bb'} > R$.

B. QUICKEST-PATH INTERDICTION MODEL

1. General Convoy Quickest-Path Interdiction Model

We employ the second decomposition algorithm developed by Israeli and Wood (2002) to solve CPIP using the CQP-CS as a sub-problem. The decomposition contains a master problem for solving the interdiction resource allocation problem and a sub-problem for finding convoy quickest-path. In its basic form, we begin with an arbitrary, resource-feasible interdiction plan. The sub-problem then reveals the convoy's quickest-path given that interdiction plan. The master problem then tries to identify a resource-feasible interdiction plan that stops the convoy from using that quickest path. (This should be trivial to accomplish in the first iteration.) Then, given that interdiction plan, the subproblem identifies a new, restricted quickest path. The master problem then tries to identify a feasible interdiction plan that stops the convoy from using either of its first two quickest paths. This process repeats until the master problem cannot identify a feasible solution that stops the convoy from using one of the paths it has responded with during the course of the algorithm. The best interdiction plan identified during the algorithm is then optimal. The decomposition algorithm follows:

Additional Definitions for Algorithm 4:

$\hat{Y}$	a subset of all arc-path incidence vectors
$\hat{\mathbf{x}}$	interdiction-plan vector derived from the master problem

$\hat{\mathbf{y}}$	path-arc incidence vector for a quickest path derived from the subproblem
$\underline{z}$	lower bound on the time for convoy to travel the quickest path
$z_{\hat{\mathbf{x}}}$	upper bound of the time for convoy to travel the quickest path
CQPP($\hat{\mathbf{x}}$)	function that finds a convoy's quickest-path given interdiction-plan $\hat{\mathbf{x}}$

Algorithm 4: A Covering Decomposition Algorithm for CPIP

Maximize the minimized convoy quickest-path by using limited interdiction resources.

Input: An instance of the convoy quickest-path problem with network $G=(N,A)$, source node *a*, sink node *b*, l_{ij}, s_{ij}, c_{ij} $\forall (i,j)\in A$, $\mathbf{r}$, and *R*.

Output: An interdiction plan $\mathbf{x}^*$ that maximizes the convoy's quickest-path

```
{
    Step 0:    Initialize: $\hat{Y} \leftarrow \varnothing$; $\underline{z} \leftarrow -\infty$;
    Step 1:    Solve the master problem:
               [MP1($\hat{Y}$)] : $z^*_{\mathrm{MP}} = \min_{\mathrm{x}} \ \mathbf{r}^{\mathrm{T}}\mathbf{x}$
                        s.t  $\hat{\mathbf{y}}^{\mathrm{T}}\mathbf{x} \ge 1 \quad \forall \hat{\mathbf{y}} \in \hat{Y}$
                             $\mathbf{x} \in \{0,1\}^{|A|}$
               If $z^*_{\mathrm{MP}} > R$ then go to Step 3;
               /* If $z^*_{\mathrm{MP}} > R$, then it is impossible to find a resource-feasible
               interdiction plan that covers all quickest paths seen in the
               algorithm so far.  Thus, the "true master problem" is infeasible. */

    Step 2:    Solve the subproblem CQPP($\hat{\mathbf{x}}$) for $\hat{\mathbf{y}}$ with objective value $z_{\hat{\mathbf{x}}}$;
               $\hat{Y} \leftarrow \hat{Y} \cup \{\hat{\mathbf{y}}\}$;
               If $\underline{z} < z_{\hat{\mathbf{x}}}$ then $\mathbf{x}' \leftarrow \hat{\mathbf{x}}$ and $\underline{z} \leftarrow z_{\hat{\mathbf{x}}}$;
               Go to Step 1;
    Step 3:    $\mathbf{x}^* \leftarrow \mathbf{x}'$; Print $\mathbf{x}^*$, $\hat{\mathbf{y}}$, $z_{\hat{\mathbf{x}}}$;
}
```

2. Enhanced Convoy Quickest-Path Interdiction Model

Israeli and Wood (2002) note that any near-optimal path from their shortest-path version of CQPP($\hat{\mathbf{x}}$) can be added to $\hat{Y}$ as long as that path's length is less than the current upper bound. That is, if the interdictor is to do as well as he knows he can, or better, he must interdict every path that is shorter than the current lower bound. He knows this is possible, but adding extra paths may make the master problem tighter and converge more quickly. Because the CQPP subproblem is solved with an implicit-enumeration algorithm, we can simply save some or all appropriate paths identified during the search, if any, in addition to the optimal path.

The enhanced algorithm, Algorithm 4E, is thus created from Algorithm 4 follows:

Algorithm 4E: An Enhanced Covering Decomposition Algorithm for CPIP

```
{
    Same as Algorithm 4 except that Step 2 is replaced by
    Step 2′:    Solve the subproblem CQPP($\hat{\mathbf{x}}$) for $\hat{\mathbf{y}}$ with objective value $z_{\hat{\mathbf{x}}}$, but
                save and order these paths identified during the course of the
                algorithm, if any:
                $\hat{Y}' = \{ \hat{\mathbf{y}}_k \mid \hat{z}_k < \underline{z}$ and ordered subject to $\hat{z}_1 < \hat{z}_2 < ... < \hat{z}_K \}$;
                $\hat{Y} \leftarrow \hat{Y} \cup \{\hat{\mathbf{y}}\}$;
                If $\underline{z} < z_{\hat{\mathbf{x}}}$ then $\mathbf{x}' \leftarrow \hat{\mathbf{x}}$ and $\underline{z} \leftarrow z_{\hat{\mathbf{x}}}$;
                If ($K > 1$) then $\hat{Y} \leftarrow \hat{Y} \cup \{\hat{\mathbf{y}}_2, \hat{\mathbf{y}}_3, ..., \hat{\mathbf{y}}_{K'}\}$, for some $K' \le K$;
    /* Because $K$ could become quite large, we simply use $K' = 2$, in practice. */
}
```

IV. COMPUTATIONAL RESULTS

This chapter presents computational results for convoy quickest-path models and their interdiction models. Tests are carried out on artificially constructed grid networks.

A. COMPUTATIONAL TEST BED

We create artificial test networks for testing purposes using these assumptions:

1) The network contains *m* rows and *n* columns of nodes, plus a source node, a sink node, an artificial source node and an artificial sink node so that $|N| = mn + 4$.

2) Each node in the $m \times n$ grid has an arc directed out of it to the "north," "south," "east" and "west," if the destination node exists in the grid; the source node has one arc directed into it from the artificial source node and *m* arcs directed out to the first column of nodes in the grid; the sink node has one arc directed out of it going to the artificial sink and *m* nodes directed into it from the last column of nodes. (See Figure 3.) Thus, $|A| = 4((m-2)(n-2)) + 3(2(m-2)+2(n-2))+2n+10$.

3) All arcs can be interdicted except for the two artificial arcs, (a', a) and (b, b'). For simplicity, interdiction of arc (i, j) in the grid does not imply interdiction of arc (j, i).

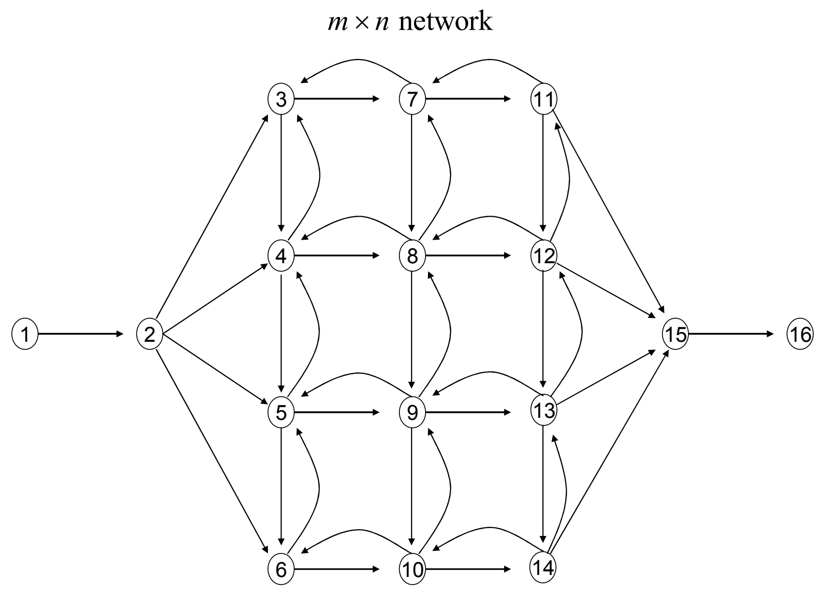


Figure 3. Example of a 4×3 test network. Nodes 1 and 16 are artificial source and sink nodes, a' and b', respectively. Node 2 is the source node *a*, while node 15 is the sink node *b*. All arcs are subject interdiction except for the artificial arcs, (1, 2) and (15, 16).

4) The convoy has a length of 42 km (400 vehicles), and the lengths of arcs are distributed randomly as: 30% have length 10 km, 30% have length 20 km, 20% have length 30 km, 10% have length 40 km and 10% have length 50 km. Effective speed limits are randomly selected, with equal probabilities, as 40 km/h, 50 km/h, 55 km/h, 60 km/h, and 70 km/h.

5) The number of lanes on all arcs is 1, or the number is selected randomly, with equal probabilities for 1 lane, 2 lanes, 3 lanes, and 4 lanes.

6) For simplicity, we assume each interdictable arc (i, j) requires exactly $r_{ij} = 1$ unit of resource to interdict.

7) All problems are solved to optimality, i.e., with a 0% relative optimality gap.

8) We code all programs using the Mosel algebraic modeling language (version 1.4.1) from Dash Optimization, and solve master problems using Dash's Xpress-MP Optimizer software (version 15.25.03). Computations are carried

out on a 2 GHz Pentium IV computer with 1 gigabyte of RAM, operating under the Windows 2000 Professional operating system.

B. CONVOY PATHS

The ability to solve CPIP will depend on how well we can solve the convoy quickest-path problem (CQPP). Therefore, we first explore the size of CQPPs that we can solve, comparing the efficiencies of the basic and enhanced algorithms, Algorithms 3 and 3E, respectively. Recall that the enhanced algorithm adds a preliminary heuristic that attempts to find a good feasible starting solution.

1. Convoy Quickest-Path

We solve a variety of problem instances and display solution times and other statistics in the following tables and figures. The listed solution times are averages of 10 trials, with numerical data generated using different random-number seeds.

To begin with, we experiment with the $m \times n$ network grid structure as n increases for fixed m; Table 2 and Figure 4 display results. The run time for Algorithm 3 seems to be increasing linearly in n while the run time for Algorithm 3E is nearly constant over the range of n evaluated. (It must increase at least linear, but these networks are not large enough to show that fact clearly.)

Network grid dimensions	Number of nodes	Number of arcs	Alg. 3 for CQP-CS		Alg. 3E for CQP-CS	
			Avg. run time (sec.)	Std. dev. (sec.)	Avg. run time (sec.)	Std. Dev. (sec.)
10×10	104	382	1.1	0.0	0.2	0.0
10×20	204	782	2.2	0.0	0.0	0.0
10×30	304	1182	4.0	0.0	0.1	0.0
10×40	404	1582	6.4	0.1	0.1	0.0
10×50	504	1982	7.9	0.1	0.1	0.0
10×60	604	2382	10.5	0.0	0.0	0.0
10×70	704	2782	10.5	0.1	0.0	0.0
10×80	804	3182	10.3	0.1	0.1	0.0
10×90	904	3582	13.1	0.0	0.0	0.0
10×100	1004	3982	15.6	0.6	0.1	0.0
10×500	5004	19982	75.3	0.7	0.2	0.0
10×1000	10004	39982	181.0	2.2	0.2	0.0
10×2000	20004	79982	414.2	1.2	0.3	0.0
10×2500	25004	99982	568.3	1.9	1.5	0.2

Table 2. A comparison of Algorithm 3 and Algorithm 3E for CQP-CS. As the value of n is increased in the $m \times n$ grid, Algorithm 3's average run time appears to increase linearly while Algorithm 3E's run time stays almost constant. This table clearly shows that the enhanced algorithm, Algorithm 3E, is much faster.

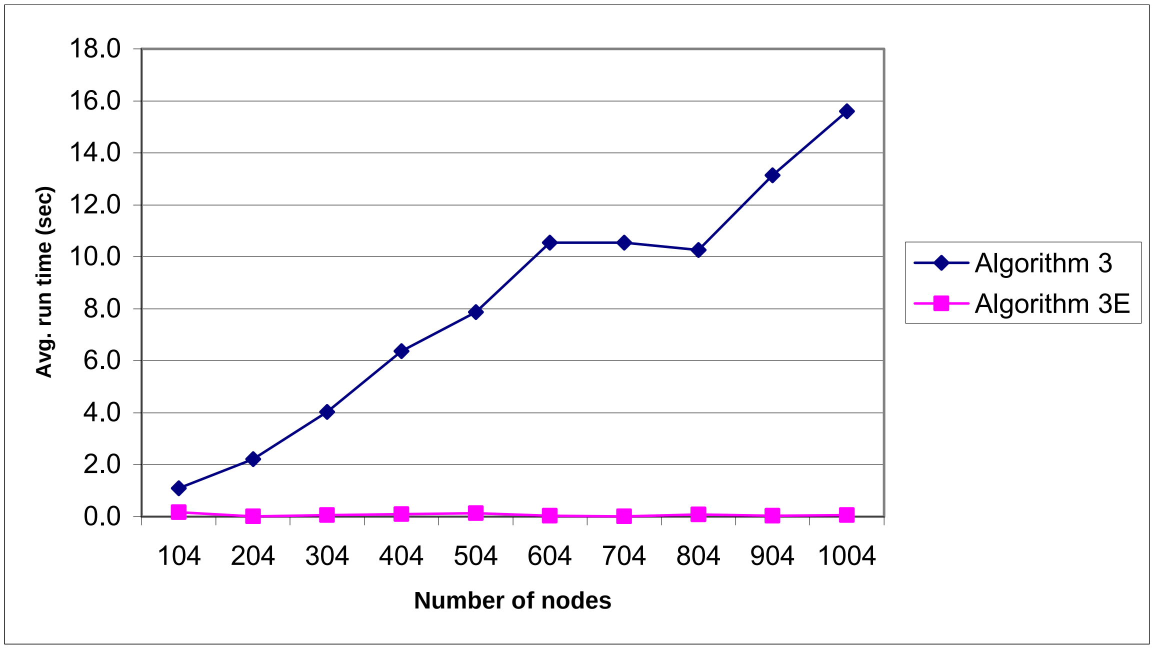


Figure 4. A plot of the data from Table 2 comparing run times for Algorithm 3 and Algorithm 3E for solving CQP-CS.

Figure 5 describes the results of an experiment in which *m* is increased for a fixed *n* in the $m \times n$ network. As *m* increases, average run times for both algorithms increase rapidly. Algorithm 3 requires more than 122 minutes, on average, to find the quickest path in the 50×10 network while only requiring 7.9 seconds in the similarly sized 10×50 network. Algorithm 3E requires more than 31 minutes, on average, to find the quickest path in the 60×10 network while requiring less than 0.01 seconds in the 10×60 network. Thus, it appears that networks that are narrow in the general direction of travel can be much easier to solve than “broad networks.”

Network grid dimensions	Number of nodes	Number of arcs	Alg. 3 for CQP-CS		Alg. 3E for CQP-CS	
			Avg. run time (sec.)	Std. dev. (sec.)	Avg. run time (sec.)	Std. dev. (sec.)
10×10	104	382	1.1	0.0	0.2	0.0
20×10	204	762	32.8	1.1	1.9	0.0
30×10	304	1142	1005.0	1.6	7.1	0.0
40×10	404	1522	1261.9	2.9	15.3	0.0
50×10	504	1902	7344.9	2.6	74.8	0.1
60×10	604	2282	-	-	1913.2	1.8

Table 3. A comparison of Algorithm 3 and Algorithm 3E for solving CQP-CS. This table shows that average run times increase superlinearly as m increases, for fixed n, in the $m \times n$ network. Comparable times for comparably sized $n \times m$ networks are much shorter. The table also clearly shows that Algorithm 3E is faster than the Algorithm 3.

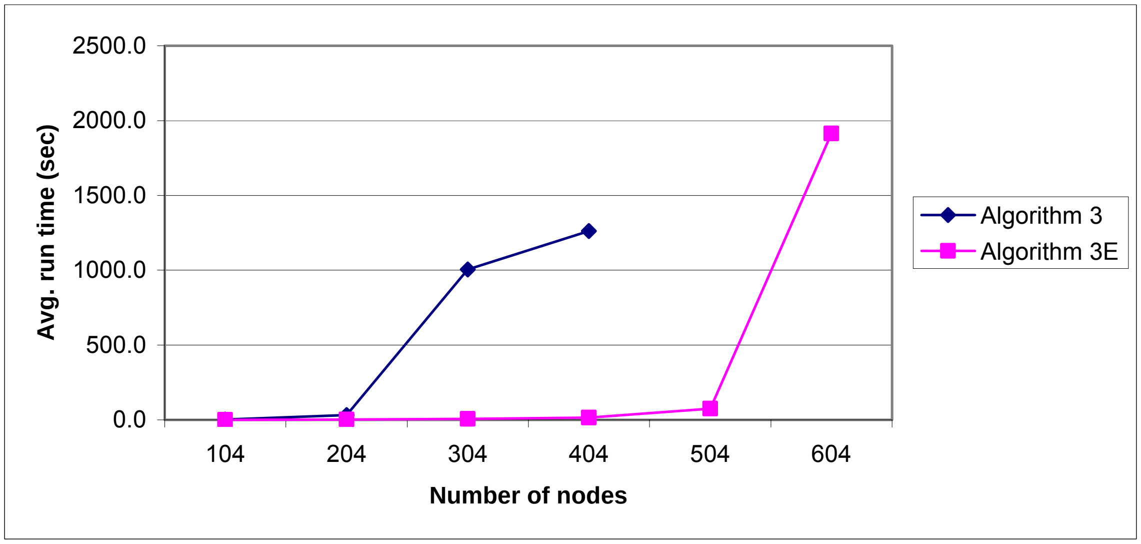


Figure 5. A plot of the data from Table 3 comparing run times for Algorithm 3 and Algorithm 3E for solving CQP-CS.

Table 4 and Figure 6 depict results for an experiment in which n increases for a square, $n \times n$ grid. Average run times for both algorithms increase quickly in $|N|$.

Network grid dimensions	Number of nodes	Number of arcs	Alg. 3 for CQP-CS		Alg. 3E for CQP-CS	
			Avg. run time (sec.)	Std. dev. (sec.)	Avg. run time (sec.)	Std. dev. (sec.)
10×10	104	382	1.1	0.0	0.2	0.0
20×20	404	1562	77.8	1.0	0.6	0.0
25×25	629	2452	437.2	1.3	10.5	0.1
30×30	904	3542	2358.4	1.9	27.3	1.8
40×40	1604	6322	-	-	642.8	5.5
50×50	2504	9902	-	-	1042.8	3.9
55×55	3029	11992	-	-	4703.8	2.4
60×60	3604	14282	-	-	5292.5	3.1

Table 4. A comparison of both Algorithm 3 and Algorithm 3E for the CQP-CS model (see also Figure 6). This table shows average run times for square grid networks. These times increase rapidly in $|N|$. Again, Algorithm 3E appears to be the faster of the two algorithms.

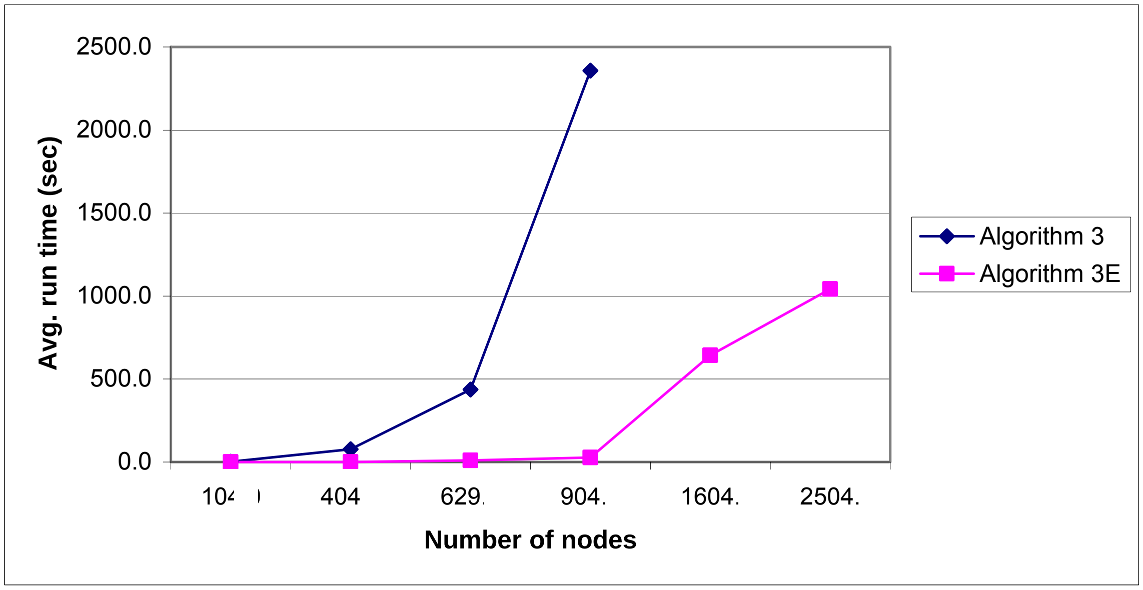


Figure 6. A plot of the data from Table 4 comparing Algorithm 3 and Algorithm 3E for CQP-CS.

All experiments clearly show that the Algorithm 3E is much more efficient than Algorithm 3. Therefore, we use Algorithm 3E for solving all CQPPs in the rest of this thesis.

2. Comparison of Convoy Quickest-Path Models

This section demonstrates that, in fact, the CQP-CS model describes convoy movement more realistically than do the shortest-path and SCTT (simple continuum traffic theory) alternatives. We can verify that difference do arise between the models by comparing path-transit times and actual "quickest" paths.

We first compare our CQP-CS model to the CQP-SCTT model suggested by Israeli et al. (2004), assuming the convoy moves in a single lane of traffic; see Table 5. (Note: The CQP-SCTT model is solve with a variant of Algorithm 3E.) Only three of the ten quickest-paths are the same for the two models. On average, the CQP-CS quickest path is 0.93 hours longer (8.67%) than the CQP-SCTT quickest path. As should be expected, all transit times for the CQP-CS model are at least as long as those for CQP-SCTT. The table also gives the "true" transit time for the quickest path identified by the SCTT algorithm. This time does not differ much from true transit time for the quickest

CS path, indicating that little error would be incurred by using the SCTT procedure. However, Table 9 will provide an instance where this is not the case.

Network grid dims.	Num. of nodes	Num. of arcs	Transit time (hours)		Same path?	Diff. (%)	"True" transit time for path A (hours)
			CQP-SCTT, single-lane (path A)	CQP-CS (path B)			
10×10	104	382	4.47	4.66	No	4.25	4.75
20×10	204	762	7.07	7.60	Yes	7.50	7.50
10×30	304	1182	4.25	4.43	No	4.24	4.58
40×10	404	1522	14.80	16.66	No	12.57	17.01
20×20	404	1562	7.99	8.50	Yes	6.38	8.50
25×25	629	2452	9.42	10.37	Yes	10.08	10.37
20×40	804	3162	6.88	7.44	No	8.14	7.82
30×35	1054	4142	10.58	11.58	No	9.45	11.61
30×50	1504	5942	11.17	12.29	No	10.03	12.35
50×50	2504	9902	17.26	19.69	No	14.08	20.40

Table 5. Comparison of CQP-SCTT and CQP-CS models in single-lane networks. This table gives the transit times, indicates whether or not the quickest paths are the same, and specifies the percentage difference in transit times, specifically 100%×(CS time – SCTT time)/(SCTT time). The average difference in transit times is about 0.93 hours (8.67%). Three of the ten quickest-paths are the same for the two models. In all cases, the "true" transit time for the SCTT quickest path differs only a little from the "true" quickest path's transit time.

Table 6 shows the results of an experiment that compares the CQP-SP model to the CQP-CS model in which the convoy moves in a single lane of traffic. CQP-SP assumes that a single vehicle moves from *a* to *b*, but adds a factor to estimate the clearing time for the convoy when its head reaches *b*. Two of the ten quickest-paths are the same for the two models, but the transit times are always greater for CQP-CS. On average, the CQP-CS quickest path is 1.24 hours (13.04%) longer than the CQP-SP quickest path.

Network grid dimensions	Number of nodes	Number of arcs	Transit time (hours)		Same path?	Diff. (%)
			CQP-SP	CQP-CS		
10×10	104	382	4.09	4.66	No	13.94
20×10	204	762	6.78	7.60	Yes	12.09
10×30	304	1182	4.00	4.43	No	10.75
40×10	404	1522	14.45	16.66	No	15.29
20×20	404	1562	7.78	8.50	No	9.25
25×25	629	2452	9.18	10.37	No	12.96
20×40	804	3162	6.53	7.44	Yes	13.94
30×35	1054	4142	10.29	11.58	No	12.54
30×50	1504	5942	10.88	12.29	No	12.96
50×50	2504	9902	16.87	19.69	No	16.72

Table 6. Comparison of CQP-SP and CQP-CS models. Two of the ten quickest-paths are the same for the two models, but the transit time is always greater for CQP-CS. The average difference in transit times, CS time − SP time, is about 1.24 hours (13.04% of the mean transit time for CQP-SP).

Table 7 compares the CQP-CS and CQP-CSML models. This should show the improvement in transit time, if any, that can be achieved by having the convoy use multiple traffic lanes. The table shows that four of the ten quickest-paths are the same for the two models, but the transit time is always less for CQP-CSML (it cannot be longer). On average, the multi-lane transit time is 0.73 hours (7.54%) faster than the single-lane transit time. Note that the largest difference in transit times is quite large, however, 26.83%. This experiment shows that using multiple lanes can definitely speed up the movement of a convoy.

Network grid dimensions	Number of nodes	Number of arcs	Transit time (hours)		Same path?	Diff. (%)
			CQP-CS	CQP-CSML		
10×10	104	382	4.66	4.32	Yes	7.87
20×10	204	762	7.60	7.37	Yes	3.12
10×30	304	1182	4.43	4.21	Yes	5.23
40×10	404	1522	16.66	15.80	No	5.44
20×20	404	1562	8.50	8.26	No	2.91
25×25	629	2452	10.37	9.79	No	5.92
20×40	804	3162	7.44	7.00	No	6.29
30×35	1054	4142	11.58	10.99	Yes	5.37
30×50	1504	5942	12.29	9.69	No	26.83
50×50	2504	9902	19.69	18.51	No	6.37

Table 7. Comparison of the CQP-CS and CQP-CSML models, i.e., constant inter-vehicle spacing using a single lane or multiple lanes, respectively. The average difference in transit time between the models, CS time − CSML time, is about 0.73 hours (7.54% of the mean transit time for CQP-CSML). Four of the ten quickest-paths are the same for the two models, but the transit time for CQP-CS is always greater than for CQP-CSML (it cannot be less).

Table 8 displays results of an experiment that compares a multi-lane version of CQP-SP, denoted CQP-SPML, to the CQP-CSML. CQP-SPML is created as a multi-lane, heuristic variant of CQP-SP by simply replacing arc speed, s_{ij} with $s_{ij}c_{ij}$. Clearing time is estimated in the same way in the two models except that s' becomes $s_{ib}c_{ib}$ for CQP-SPML. With one exception, all quickest paths differ between the two models; transit time is always shorter for CQP-CSML (it cannot be longer). On average, the CQP-SPML transit time is 4.22 hours (78.48%) shorter than the transit time for CQP-SP.

Network grid dimensions	Number of nodes	Number of arcs	Transit time (hours)		Same path?	Diff. (%)
			CQP-SPML	CQP-CSML		
10×10	104	382	3.06	4.32	No	41.18
20×10	204	762	3.73	7.37	Yes	97.59
10×30	304	1182	2.76	4.21	No	52.54
40×10	404	1522	8.14	15.80	No	94.10
20×20	404	1562	4.50	8.26	No	83.56
25×25	629	2452	5.03	9.79	No	94.63
20×40	804	3162	3.62	7.00	No	93.37
30×35	1054	4142	5.05	10.99	No	117.62
30×50	1504	5942	6.69	9.69	No	44.84
50×50	2504	9902	11.19	18.51	No	65.42

Table 8. Comparison of CQP-SPML and CQP-CSML models. The average difference in transit times, CSML time − SPML time, is about 4.22 hours (78.48% of the mean transit time for CQP-SP). With one exception, the quickest paths differ in all trials.

Table 9 compares convoy quickest paths derived from the CQP-SCTT and CQP-CSML models. On average, the CSML transit time is 0.64 hours (6.31%) longer than the SCTT transit time. Nine of the ten problems yield different quickest paths for the two models. Transit times for CQP-CS are longer except in one case, the 30×50 grid network. That instance can be explained as follows: Both models are slowed down by an unavoidable, low-speed arc in the first part of their quickest paths. Further along, CSML can take advantage of an arc with four high-speed lanes, while SCTT cannot.

Table 9 also shows that all SCTT quickest paths but one have true transit times that are very close to the true, CS quickest-path transit time. This means that, except in one case, one would not make a large error by using the SCTT procedure to find a quickest path. However, for the one exception, the 30×50 grid, the "true" (CS) quickest path is 100% × (11.63–9.69)/9.69 = 20.0% shorter than the SCTT-identified quickest path. This seems like a large error that cannot be ignored.

Network grid dims.	Num. of nodes	Num. of arcs	Transit time (hours)		Same path?	Diff. (%)	"True" transit time for path A (hours)
			CQP-SCTT (path A)	CQP-CSML (path B)			
10×10	104	382	4.26	4.32	No	1.41	4.45
20×10	204	762	6.78	7.37	Yes	8.70	7.37
10×30	304	1182	4.04	4.21	No	4.21	4.56
40×10	404	1522	14.53	15.80	No	8.74	15.98
20×20	404	1562	7.92	8.26	No	4.29	8.37
25×25	629	2452	9.21	9.79	No	6.30	9.97
20×40	804	3162	6.88	7.00	No	1.74	7.02
30×35	1054	4142	10.37	10.99	No	5.98	11.00
30×50	1504	5942	11.10	9.69	No	–14.55	11.63
50×50	2504	9902	17.26	18.51	No	7.24	19.28

Table 9. Comparison of CQP-SCTT and CQP-CSML. The average difference in transit time, CSML time – SCTT time, is about 0.64 hours (6.31% of the mean transit time for CQP-SCTT). All quickest paths differ except in one trial, and all transit times for the CQP-CSML model exceed those for CQP-SCTT except in one case. That last column evaluates the "true," CSML transit time for path A, which was identified as the quickest path by SCTT. This value is not too different from the CSML quickest-path time in most cases, except for the 30×50 grid where these values differ by 100% × (11.63–9.69)/9.69 = 20.0%.

The experiments described above show that the shortest-path and SCTT approximations of the quickest path can differ substantially from the "true" quickest path. We have argued how the CQP-CSML model must be more accurate than the alternatives, and we see that it is consistently more conservative. Until even more accurate models are devised, we believe that CQP-CSML should be used, and we will use it within our algorithms for solving the convoy-path interdiction problem.

C. CONVOY PATH INTERDICTION

This section explores the efficiency of the basic and enhanced algorithms, Algorithm 4 and 4E, respectively, for solving CPIP. We use Algorithm 3E to solve the CQP-CSML subproblems in all cases. Recall that the enhanced interdiction algorithm, Algorithm 4E, can add more than one constraint per iteration to the CPIP master problem, whereas Algorithm 4 will always add just one.

1. Convoy Quickest-Path Interdiction with Constant Inter-vehicle Spacing and Multiple Lanes (CPIP-CSML)

We solve a variety of network problems with Algorithm 4 and Algorithm 4E and display solution times in the tables below. Recall that Algorithm 4E simply employs $K' = 2$, which means that that up to two constraints are added to the master problem in each iteration rather than the single constraint of the basic algorithm. Algorithm 4E may be able to improve run times compared to Algorithm 4 if, in at least one iteration, it finds two paths whose transit times are less than the current upper bound on a post-interdiction transit time. The listed solution times are average values of 10 trials using different random-number seeds.

We experiment with the $m \times n$ network structure as n increases for fixed m; Table 10 displays results. It appears that increases in the size of the network and in the amount of interdiction resource both increase run times. The table shows that Algorithm 4E is faster than Algorithm 4.

Network grid dims.	Interdict. resource	Alg. 4 for CPIP-CSML		Alg. 4E for CPIP-CSML		Multiple optimal interdiction solutions?
		Avg. run time (sec.)	Std.dev. (sec.)	Avg. run time (sec.)	Std.dev. (sec.)	
10×10	2	1.4	0.0	1.4	0.0	No
	3	2.7	0.1	2.4	0.0	No
	4	4.2	0.2	3.9	0.1	No
10×20	4	2.0	0.1	2.0	0.1	Yes
	5	2.2	0.1	2.2	0.1	Yes
	6	3.9	0.3	3.7	0.3	Yes
10×30	4	3.7	0.4	3.4	0.2	Yes
	5	6.8	0.9	6.3	0.9	Yes
	6	7.7	1.3	7.6	1.5	Yes
10×40	4	4.6	0.5	4.5	0.6	Yes
	5	7.5	1.4	6.2	0.7	Yes
	6	12.2	1.0	11.6	1.7	Yes

Table 10. Comparison of Algorithms 4 and 4E for CPIP-CSML as *n* increases in an $m \times n$ grid network with *m* fixed. Algorithm 4E is clearly faster than Algorithm 4.

Table 11 displays results for an experiment in which *m* increases in the $m \times n$ network given fixed *m*. As *m* increases, the run times for both algorithms increase rapidly as do the standard deviations; run times also increase with increasing interdiction resource. As above, we clearly see that Algorithm 4E is faster than Algorithm 4.

Network grid dims.	Interdict. resource	Alg. 4 for CPIP-CSML		Alg. 4E for CPIP-CSML		Multiple optimal interdiction solutions?
		Avg. run time (sec.)	Std.dev. (sec.)	Avg. run time (sec.)	Std.dev. (sec.)	
10×10	2	1.4	0.0	1.4	0.0	No
	3	2.7	0.1	2.4	0.0	No
	4	4.2	0.2	3.9	0.1	No
20×10	2	8.9	1.8	8.9	1.8	Yes
	3	52.9	1.3	48.7	1.9	No
	4	83.3	5.5	76.7	1.8	No
30×10	2	704.8	24.1	647.0	19.5	Yes
40×10	2	2521.5	121.1	2258.5	101.6	Yes

Table 11. Comparison of Algorithm 4 and Algorithm 4E for CPIP-CSML as *m* increases in an $m \times n$ grid network with *n* fixed. Average run times increase much more rapidly in these "broad networks" than they do in the "narrow networks" of Table 10. Run times also increase with increasing interdiction resource. As in the previous experiment, Algorithm 4E is clearly faster than Algorithm 4.

Table 12 presents results for an experiment in *n* increases for a square $n \times n$ grid. Run times for both algorithms appear to increase rapidly both as a function of *n* and as function of the amount of interdiction resource. Again, Algorithm 4E is clearly faster than Algorithm 4.

Network grid dims.	Interdict. resource	Alg. 4 for CPIP-CSML		Alg. 4E for CPIP-CSML		Multiple optimal interdiction solutions?
		Avg. run time (sec.)	Std.dev. (sec.)	Avg. run time (sec.)	Std.dev. (sec.)	
10×10	2	1.4	0.0	1.4	0.0	No
	3	2.7	0.1	2.4	0.0	No
	4	4.2	0.2	3.9	0.1	No
15×15	3	11.5	1.3	10.3	1.1	Yes
	4	17.8	1.9	16.7	2.8	Yes
	5	34.0	7.0	31.7	5.2	Yes
20×20	4	44.7	18.2	26.6	4.0	Yes
	5	57.2	11.0	54.2	19.4	Yes
	6	297.6	33.5	237.8	27.1	Yes
25×25	2	45.8	1.8	43.3	0.5	Yes
	3	137.9	1.6	136.9	3.6	No
	4	363.3	89.9	359.7	46.2	Yes

Table 12. Comparison of Algorithm 4 and Algorithm 4E for CPIP-CSML. This table describes an experiment in which n increases in an $n \times n$ grid network. Once again, run times increase rapidly with increasing network size, and increasing interdiction resource. Again, Algorithm 4E is somewhat faster than Algorithm 4.

Algorithm 4E nominally uses $K' = 2$, which is the maximum number of constraints that can be added in each iteration. Algorithm 4E is better than Algorithm 4, so it may be possible to improve performance by increasing K' beyond 2. Table 13 compares run times as K' is varied between 2 and 5. The table clearly shows that $K' = 2$ provides the best average run time, except in the case of the 20×20 grid with interdiction resource set at 6.

Ignoring random effects, using $K' > 2$ can reduce solution times only if it reduces the number of main iterations required in Algorithm 4E. Table 14 shows that it does sometimes, but not consistently. Furthermore, even if the number of iterations is reduced, this cannot guarantee a reduced run time because increased overhead.

Network grid dims.	Interdict. resource	Alg. 4		Algorithm 4E							
				$K'=2$		$K'=3$		$K'=4$		$K'=5$	
		Avg. run time (sec.)	Avg. Iter.	Avg. run time (sec.)	Avg. Iter.	Avg. run time (sec.)	Avg. Iter.	Avg. run time (sec.)	Avg. Iter.	Avg. run time (sec.)	Avg. Iter.
10×10	5	5.4	23.0	4.9	20.0	4.9	20.0	4.9	20.0	4.9	20.0
	6	8.6	34.0	7.6	31.0	9.3	38.0	9.2	38.0	9.1	38.0
	7	13.3	49.0	10.3	48.0	11.8	48.0	11.7	48.0	11.8	48.0
15×10	5	32.5	22.4	35.3	34.5	37.1	34.0	36.6	34.0	35.1	34.0
	6	94.5	77.8	89.6	76.8	100.4	86.0	97.2	83.4	93.3	82.2
	7	150.6	129.4	122.0	116.0	155.4	125.4	147.7	116.4	139.4	103.2
10×20	5	2.3	11.0	2.3	11.0	2.3	11.0	2.3	11.0	2.3	11.0
	6	3.8	16.6	3.7	16.0	3.8	16.0	3.8	16.0	3.8	16.0
	7	5.3	21.8	5.1	20.4	5.1	20.4	5.1	20.4	5.1	20.4
15×17	4	49.1	25.5	45.5	24.0	45.6	24.8	45.6	22.8	45.7	21.9
	5	63.8	33.7	55.2	29.1	57.4	29.8	61.3	31.7	64.1	32.5
	6	111.2	60.9	81.0	45.6	91.6	48.0	88.0	46.0	90.0	46.8
20×20	4	44.7	16.5	26.6	14.6	36.8	15.1	26.5	15.3	30.8	15.3
	5	57.2	24.9	54.2	24.8	58.5	25.0	58.9	26.0	57.4	25.0
	6	297.6	70.1	237.8	68.3	227.7	64.1	209.6	62.1	185.7	65.5

Table 13. Comparison of run times solving CPIP-CS, for Algorithm 4 and Algorithm 4E as K' varies. Note that all standard deviations for run times (not shown) are roughly 10% of the average run times. This table shows that Algorithm 4E, with $K'=2$, can improve run times over Algorithm 4 by reducing the number of master-problem iterations. Increasing K' beyond 2 can further reduce the number of iterations, and sometimes reduce run times, but the overhead must become too great for reduced iterations to guarantee a reduced run time.

D. A POTENTIAL IMPROVEMENT FOR ALGORITHMS 3 AND 3E

At the end of this research effort, we realized that Test 1 in Algorithm 3 and 3E could be strengthened. If we compute backward distances from *b* starting at 0 rather than at t' (recall that t' is a lower bound on the clearing time once the convoy's head reaches *b*), then Test 1 looks like:

$$\text{if}\,(\,F(i)+t_{ij}+t'+d(j)<UB \text{ and } \textit{not}\ \text{isOnPath}(j)\,)\ \{\quad \text{/* Test 1a */.}$$

If we define t_i'' as the amount of time that a single vehicle at the tail of the convoy would require to reach i given that the head of the convoy is at i (and the body of the convoy lets this vehicle pass without hindrance!), then another valid test would be

if ($F(i)+t_{ij}+t_i''+d(j)<UB$ and *not* isOnPath(j)) { /* Test 1b */.

It follows that the following test is valid and stronger than either Test 1a or Test 1b:

if ($F(i)+t_{ij}+\max\{t',t_i''\}+d(j)<UB$ and *not* isOnPath(j)) { /* Test 1c */.

Table 14 repeats part of Table 4 to explore the value of Test 1c. We denote the potentially improved version of Algorithm 3E as Algorithm 3E′ and compare it to Algorithm 3E on some $n\times n$ grids. The results are clearly promising and indicate that larger problems instances of CPIP than we have been able to solve will, in fact, be solvable. And, even stronger versions of Test 1c can be defined: Only additional testing will be able to determine the extent of improvements that are possible.

Network grid dims.	Number of nodes	Number of arcs	Alg. 3E for CQP-CS		Alg. 3E′ for CQP-CS	
			Avg. run time (sec.)	Std. dev. (sec.)	Avg. run time (sec.)	Std. dev. (sec.)
10×10	104	382	0.2	0.0	0.0	0.0
20×20	404	1562	0.6	0.0	0.0	0.0
25×25	629	2452	10.5	0.1	0.3	0.0
30×30	904	3542	27.3	1.8	0.8	0.0
40×40	1604	6322	642.8	5.5	23.2	0.4
50×50	2504	9902	1042.8	3.9	36.8	1.8

Table 14. Comparison of Algorithm 3E for CQP-CS with Algorithm 3E′, which uses a stronger version of Test 1. The table shows that the stronger test substantially reduces computation time. This bodes well for solving larger instances of CPIP using our decomposition methodology.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

This thesis devised a method to solve "the convoy-path interdiction problem" (CPIP) in which an interdictor uses limited resources to attack and disrupt road segments ("arcs") or road intersections ("nodes") in a road network in order to delay an adversary's convoy from reaching its destination. The convoy moves between a known origin node *a* and destination node *b* using a "quickest path." The quickest *a*-*b* path requires the least amount of time for the full convoy to move from *a* to *b*, and takes into account the size or length of the convoy, and each road segment's capacity (effective speed limit) and length. Because a convoy can be quite long and must move according to doctrine, this "convoy quickest path problem" (CQPP) is more complicated than a standard shortest-path problem.

The analysis is divided to two parts. The first part devises an algorithm to identify a quickest *a*-*b* path and determine that path's transit time. The second describes an algorithm to find an optimal allocation of limited interdiction resources to maximize the post-interdiction transit time of a quickest path.

In the first part, we have reviewed one existing quickest path model, "CQP-SCTT," and have devised a simple extension of the standard shortest-path, "CQP-SP." The first model is based on simple continuum traffic theory and violates our doctrinal assumption that the convoy will maintain constant inter-vehicle spacing. We find that neither model is appropriate for describing such a convoy, and confirm this through later computational testing. Therefore, we have developed the CQP-CS model (convoy quickest-path with constant spacing). This model assumes that vehicles in the convoy move in a single lane of traffic and maintain constant inter-vehicle spacing; thus, the speed of the convoy at any moment is dictated by the slowest speed limit that any vehicle is currently subject to.

To solve CQP-CS, we have developed an implicit path-enumeration algorithm (a type of "A*-search" or "branch-and-bound algorithm"), and have enhanced it as Algorithm 3E, which uses a heuristic to find a good, initial, feasible solution. Algorithm 3E requires less than 20 minutes to solve CQP-CS on a 50×50 grid network (2504 nodes and 9902 arcs). We also extend the CQP-CS model to CQP-CSML, which models the convoy's use of multiple and varying numbers of traffic lanes.

In second part of this thesis, we have used the set-covering decomposition and algorithm of Israeli and Wood (2002), Algorithm 4, to identify an optimal interdiction plan. We have also developed an enhanced version, Algorithm 4E, which tightens the master problem and converges more quickly. Algorithm 4 requires about 6 minutes to solve a 25×25 grid network problem (629 nodes and 2425 arcs) when the interdictor can interdict four arcs. Algorithm 4E improves on the run time of Algorithm 4 by 10.8%, on average, with a maximum improvement of 68%.

LIST OF REFERNCES

Ahuja,R.K.,Magnanti,T.L. and Orlin,J.B. (1993). *Network Flows*, Prentice Hall, Englewood Cliffs, New Jersey.

Bazarra, M.S., Sherali, H.D. and Shetty, C.M. (1993). *Nonlinear Programming: Theory and Algorithms,* John Wiley and Sons, New York.

Ball, M.O., Golden, B.L. and Vohra, R.V. (1989). "Finding the most vital arcs in a network," *Operations Research Letters*, **8**, pp. 73-76.

Caldwell T. (1961). "On finding minimal routes in a network with turning penalties," *Communications of the ACM*, **4**, 107-108.

Fulkerson, D.R. and Harding, G.C. (1977). "Maximizing the minimum source-sink path subject to a budget constraint," *Mathematical Programming,* **30**, pp. 116-118.

Golden, B. (1978). "A problem in network interdiction," *Naval Research Logistics Quarterly,* **25**, 711-713.

Israeli, E. and Wood, K. (2002). "Shortest path network interdiction," *Networks,* **40**, 97-111.

Israeli, E., Wevley, C., and Wood, K. (2004). "The quickest path network interdiction problem," Working paper, Naval Postgraduate School, Monterey, California, February.

Kuhne, R., and Michalopoulous, P. (1998). "Continuum Flow Models," *Revised Monograph on Traffic Flow Theory*, 5(3,Whole No. 165). Available FTP: Hostname: tfhrc.gov Directory: its/tft/tft.htm

Lighthill, M. H. and Whitham, G. B. (1955). "On Kinematic Waves-II: A Theory of Traffic Flow on Long Crowded Roads," *Proceedings, Royal Society* (London), A229, No. 1178, 317-345.

Malik, K., Mittal, A.K., and Gupta, S.K. (1989). "The k-most vital arcs in the shortest path problem," *Operations Research Letters*, **8**, 223-227

Held, H., Wood, K., and Woodruff, D. (2004). "Delaying an adversary in a stochastic network," working paper.

Russell, S. and Norvig, P., (1995). *Artificial Intelligence: A Modern Approach*, Prentice Hall, Saddle River, New Jersey.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor R. Kevin Wood
   Naval Postgraduate School
   Monterey, California

4. Professor Johannes O. Royset
   Naval Postgraduate School
   Monterey, California

5. Jung Woo Yim
   Naval Postgraduate School
   Monterey, California

6. Ki Hwan Kim
   Naval Postgraduate School
   Monterey, California

7. Si Won Park
   Naval Postgraduate School
   Monterey, California